# Introduction to dplyr

Randall Pruim

Big Data Ignite 2016

# Tidy (Rectangular) Data

**General Form**

- rows = cases/observational units
- columns = variables
- no exceptions

**For plotting**

- row for each mark on the plot (cases = marks)
- columns contain information needed to determine position, color, size, shape, etc.

**Many other reasons to want tidy data**

# Other names for rectangular data

- tabular data
- data table
- data frame
- tibble

Note:

- In R, a data frame is a particular class of container used to store this sort of data.
- `dplyr` functions produce tibbles which can be data frames plus some additional information or an abstraction of a data base connection.

# Data Verbs

`dplyr` works primarily by defining a set of **data verbs**

**A data verb is a function that**

- takes data table as its first argument
- returns a data table

Data table may be a data frame or a tibble.

# Five Main Data Verbs

Data verbs take data tables as input and give data tables as output

1. `filter()`: subsets *cases* (i.e. rows)
2. `select()`: subsets *variables* (i.e. columns)
   - `matches()`, `starts_with()`, `ends_with()`
3. `mutate()`: creates new variables
4. `arrange()`: reorders the cases
5. `summarize()`: reduce to a single row (summary stats)
   - `n()`: number of observations in the current group

# Chaining Syntax

The pipe syntax (%>%) provides an alternative syntax that works well for sequential operations on data.

- `x %>% f(y)` is the same as `f(x, y)`
- `y %>% f(x, ., z)` is the same as `f(x, y, z)`

Read %>% as "then"

```
# do verb1, then verb2, then verb3
data %>%
  verb1(arguments1) %>%
  verb2(arguments2) %>%
  verb3(arguments3)
```

# Little Bunny Foo Foo

*Little bunny Foo Foo*
*Went hopping through the forest*
*Scooping up the field mice*
*And bopping them on the head.*

# Chaining

### Foo Foo without chaining

```
bop(
  scoop(
    hop(foo_foo, through = forest),
    up = field_mice),
  on = head )
```

# Chaining

### Foo Foo with chaining

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mouse) %>%
  bop(on = head)
```

### Foo Foo without chaining

```
bop(
  scoop(
    hop(foo_foo, through = forest),
    up = field_mice),
  on = head )
```

# Time for Some Examples

This will give you access to the data used in the examples below.

```
require(babynames)
require(NHANES)
require(nycflights13)
Babynames <- babynames  # change to capitalized version
```

# Babynames

Number of US children given each name (minimum five children) each year since 1880. [1.8 M rows; 337.1 M kids]

```
Babynames %>% head()
```

```
# A tibble: 6 × 5
   year   sex      name     n       prop
  <dbl> <chr>     <chr> <int>      <dbl>
1  1880     F      Mary  7065 0.07238359
2  1880     F      Anna  2604 0.02667896
3  1880     F      Emma  2003 0.02052149
4  1880     F Elizabeth  1939 0.01986579
5  1880     F    Minnie  1746 0.01788843
6  1880     F  Margaret  1578 0.01616720
```

# NHANES Data

Roughly equivalent to a random sample of 10,000 Americans.

- Actually created by a weighted random sampling from raw data

```r
require(NHANES)
names(NHANES)
```

```
 [1] "ID"             "SurveyYr"        "Gender"         "Age"            "AgeDecade"
 [6] "AgeMonths"      "Race1"           "Race3"          "Education"      "MaritalStatus"
[11] "HHIncome"       "HHIncomeMid"     "Poverty"        "HomeRooms"      "HomeOwn"
[16] "Work"           "Weight"          "Length"         "HeadCirc"       "Height"
[21] "BMI"            "BMICatUnder20yrs" "BMI_WHO"       "Pulse"          "BPSysAve"
[26] "BPDiaAve"       "BPSys1"          "BPDia1"         "BPSys2"         "BPDia2"
[31] "BPSys3"         "BPDia3"          "Testosterone"   "DirectChol"     "TotChol"
[36] "UrineVol1"      "UrineFlow1"      "UrineVol2"      "UrineFlow2"     "Diabetes"
[41] "DiabetesAge"    "HealthGen"       "DaysPhysHlthBad" "DaysMentHlthBad" "LittleInterest"
[46] "Depressed"      "nPregnancies"    "nBabies"        "Age1stBaby"     "SleepHrsNight"
[51] "SleepTrouble"   "PhysActive"      "PhysActiveDays" "TVHrsDay"       "CompHrsDay"
[56] "TVHrsDayChild"  "CompHrsDayChild" "Alcohol12PlusYr" "AlcoholDay"    "AlcoholYear"
[61] "SmokeNow"       "Smoke100"        "Smoke100n"      "SmokeAge"       "Marijuana"
[66] "AgeFirstMarij"  "RegularMarij"    "AgeRegMarij"    "HardDrugs"      "SexEver"
[71] "SexAge"         "SexNumPartnLife" "SexNumPartYear" "SameSex"        "SexOrientation"
[76] "PregnantNow"
```

# NYC flights

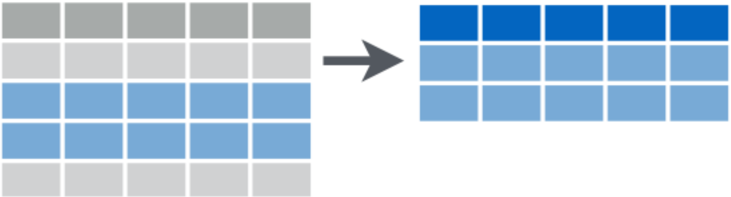
Several data tables providing information on all US flights into and out of JFK, EWR, and LGA in 2013. (Part of a larger data base that includes all US flights since 1987.)

```
require(nycflights13)
names(flights)
```

```
 [1] "year"           "month"          "day"
 [4] "dep_time"       "sched_dep_time" "dep_delay"
 [7] "arr_time"       "sched_arr_time" "arr_delay"
[10] "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"
[16] "distance"       "hour"           "minute"
[19] "time_hour"
```

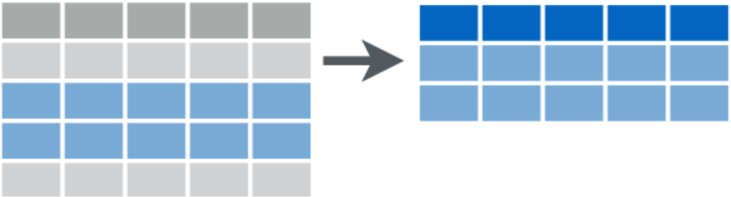
Other data sets: `airlilnes`, `planes`, `airports`, `weather`

# 1. filter(): subsets *cases*



```
CollegeAge <-
  NHANES %>% filter(Age %in% 18:22)
Smokers <-
  NHANES %>% filter(SmokeNow == "Yes")
```

# 1. filter(): subsets *cases*



```r
CollegeSmokers <-
  NHANES %>%
  filter(Age %in% 18:22) %>%
  filter(SmokeNow == "Yes")
CollegeSmokers2 <-
  NHANES %>% filter(Age %in% 18:22, SmokeNow == "Yes")
AttendedCollege <-
  NHANES %>%
  filter(Education %in% c("Some College", "College Grad"))
```

# Variations on filter()

Like `filter()` these all return a subset of the **rows** of the data table

- `distinct()`: returns the **unique** rows in a table
- `sample_n()`: returns **random** rows (specify number)
- `sample_frac()`: returns **random** rows (specify fraction)
- `head()`: grab the **first** few rows
- `tail()`: grab the **last** few rows
- `top_n()`: top few after sorting by a variable

# Examples of filter() and friends

Because the NHANES data set was created using random sampling with replacement to mimic the probability sample used when the raw data were gathered, some of the rows are duplicates:

```
NHANES %>%
  distinct() %>%
  nrow()
```

```
[1] 7832
```

```
NHANES %>% nrow()
```

```
[1] 10000
```

# Examples of filter() and friends

`top_n()` is useful for finding the extremes in the data

```
Babynames %>% top_n(2, prop)
```

```
# A tibble: 2 × 5
   year   sex  name      n       prop
  <dbl> <chr> <chr> <int>      <dbl>
1  1880     M  John  9655 0.08154561
2  1881     M  John  8769 0.08098149
```

```
Babynames %>% top_n(2, n)
```

```
# A tibble: 2 × 5
   year   sex  name      n       prop
  <dbl> <chr> <chr> <int>      <dbl>
1  1947     F Linda 99680 0.05483648
2  1948     F Linda 96205 0.05520789
```

# Bottom's up

```r
Babynames %>% top_n(2, -n)  # trick for bottom n
```

```
# A tibble: 254,615 × 5
    year   sex      name     n         prop
   <dbl> <chr>     <chr> <int>        <dbl>
1   1880     F    Adelle     5 5.122688e-05
2   1880     F     Adina     5 5.122688e-05
3   1880     F  Adrienne     5 5.122688e-05
4   1880     F Albertine     5 5.122688e-05
5   1880     F      Alys     5 5.122688e-05
6   1880     F       Ana     5 5.122688e-05
7   1880     F  Araminta     5 5.122688e-05
8   1880     F    Arthur     5 5.122688e-05
9   1880     F    Birtha     5 5.122688e-05
10  1880     F     Bulah     5 5.122688e-05
# ... with 254,605 more rows
```

## 2. select(): subsets *variables*

## 2. select(): subsets *variables*

Variables related to questions about sleep.

```
NHANESsleep <-
  NHANES %>%
  select(Gender, Age, Weight, Race1, Race3,
         matches("Sleep"), matches("TV"))
NHANESsleep %>% names()
```

```
[1] "Gender"        "Age"           "Weight"
[4] "Race1"         "Race3"         "SleepHrsNight"
[7] "SleepTrouble"  "TVHrsDay"      "TVHrsDayChild"
```

```
NHANESsleep %>% dim()
```

```
[1] 10000     9
```

```
NHANES %>% dim()
```

```
[1] 10000    76
```

Note: `names()` and `dim()` are not data verbs so the terminate our chain of data table

# 3. mutate(): create new variables

# 3. mutate(): create new variables

```
NHANESsleep %>%
  mutate(Weightlb = Weight*2.2) %>%
  select(Weight, Weightlb) %>%
  top_n(3, Weight)     # we will get 4 because of ties
```

```
# A tibble: 4 × 2
  Weight Weightlb
   <dbl>    <dbl>
1  230.7   507.54
2  230.7   507.54
3  223.0   490.60
4  223.0   490.60
```

## 3. mutate(): create new variables

Variations

- reuse variable name to replace
- use `rename()` to rename varables
- `transmute()` only keeps variables mentioned (`mutate()` + `select()`)

```
NHANES %>%
  transmute(Weightlb = Weight * 2.2) %>%
  sample_n(2)
```

```
# A tibble: 2 × 1
  Weightlb
     <dbl>
1    61.38
2   142.12
```

# 4. arrange(): reorder the rows

```
NHANES %>%
  distinct() %>%
  mutate(Weightlb = 2.2 * Weight) %>%
  select(Age, Gender, Weightlb) %>%
  arrange(-Weightlb)
```

```
# A tibble: 7,832 × 3
     Age Gender Weightlb
   <int> <fctr>    <dbl>
1     52 female   507.54
2     37   male   490.60
3     63   male   446.60
4     25 female   437.14
5     38   male   420.42
6     33 female   415.58
7     46 female   414.70
8     30 female   412.50
9     31   male   405.90
10    34   male   399.08
# ... with 7,822 more rows
```

# 5. summarise(): 1-row summary

```r
# number of people (cases) in NHANES
NHANES %>% summarise(n())
```

```
# A tibble: 1 × 1
  `n()`
  <int>
1 10000
```

```r
nrow(NHANES)
```

```
[1] 10000
```

# 5. summarize(): 1-row summary

Can have multiple columns in our 1-row summary

```
NHANES %>%
  mutate(Weightlb = Weight * 2.2) %>%
  summarise(
    n = n(),
    mean_weight = mean(Weightlb, na.rm=TRUE),
    mean_age = mean(Age, na.rm = TRUE))
```

```
# A tibble: 1 × 3
      n mean_weight mean_age
  <int>       <dbl>    <dbl>
1 10000      156.16  36.7421
```

# Grouping

The `dplyr` tools become much more powerful in combination with grouping

- `group_by()`: successive functions are applied within groups
- `ungroup()`: remove all groups
- `groups()`: show the current grouping

# summarize() with group_by()

```
NHANES %>%
  mutate(Weightlb = Weight * 2.2) %>%
  group_by(Education) %>%
  summarise(
    n = n(), mean_weight = mean(Weightlb, na.rm=TRUE),
    mean_age = mean(Age, na.rm = TRUE)) %>%
  arrange(mean_weight) %>% data.frame()
```

```
       Education    n mean_weight  mean_age
1           <NA> 2779    91.69645  9.645916
2      8th Grade  451   173.03296 54.297118
3   College Grad 2098   176.77638 47.015729
4 9 - 11th Grade  888   180.53893 48.136261
5    High School 1517   183.10185 47.533949
6   Some College 2267   185.20643 45.273048
```

# Your Turn

When starting, it can be helpful to work with a subset of the data. When you have your data wrangling statements in working order, shift to the entire data table.

```
OldBabies <-
  Babynames %>%
  filter(year < 1885)
dim(OldBabies)
```

```
[1] 10443     5
```

```
names(OldBabies)
```

```
[1] "year" "sex"  "name" "n"    "prop"
```

# How many babies are represented?

# How many babies are represented?

```
OldBabies %>%
  summarise(total = ????)
```

# How many babies are represented?

```
OldBabies %>%
  summarise(total = ????)
```

```
OldBabies %>%
  summarise(total = sum(n))
```

```
# A tibble: 1 × 1
    total
    <int>
1 1076138
```

Note: This doesn't include babies with rare names or people who were not registered with SSA.

# How many babies are there in each year?

```
OldBabies %>%
  group_by(????) %>%
  summarise(total = ????)
```

# How many babies are there in each year?

```
OldBabies %>%
  group_by(year) %>%
  summarise(total = sum(n))
```

```
# A tibble: 5 × 2
   year  total
  <dbl>  <int>
1  1880 201484
2  1881 192699
3  1882 221538
4  1883 216950
5  1884 243467
```

# How many distinct names in each year?

```
OldBabies %>%
  group_by(????) %>%
  summarise(name_count = n_distinct(????))
```

# How many distinct names in each year?

```
OldBabies %>%
  group_by(year) %>%
  summarise(name_count = n_distinct(name))
```

```
# A tibble: 5 × 2
   year name_count
  <dbl>      <int>
1  1880       1889
2  1881       1830
3  1882       2012
4  1883       1962
5  1884       2158
```

# How many distinct names in each year?

```
OldBabies %>%
  group_by(????, ????) %>%
  summarise(????)
```

# How many distinct names?

```
OldBabies %>%
  group_by(year) %>%
  summarise(names = n(),
            distinct_names = n_distinct(name),
            duplicates = names - distinct_names)
```

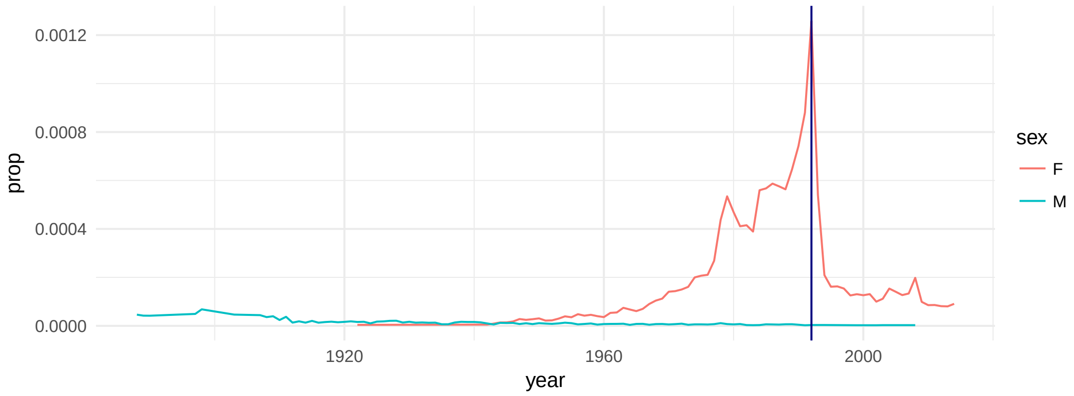
```
# A tibble: 5 × 4
   year names distinct_names duplicates
  <dbl> <int>          <int>      <int>
1  1880  2000           1889        111
2  1881  1935           1830        105
3  1882  2127           2012        115
4  1883  2084           1962        122
5  1884  2297           2158        139
```

# Track the yearly number of Hillarys.

```
Hillary <-
  Babynames %>%
  filter(name == "Hillary")
```

# Plot the results

```
Hillary %>%
  ggplot(aes(x = year, y = prop, colour = sex)) +
  geom_line() +
  geom_vline(xintercept = 1992, colour = "navy")
```



# Some Exercises for you to try

1. Plot a set of related names over time.
2. Find the year in which your name was most popular.
3. Find the largest year-over-year change in popularity for a name.
4. Look for trends in first letters of names over time.
5. Look for trends in the length of names over time. (`nchar()` is useful)

# tidyr

The `tidyr` package provides some additional data verbs, including

- `gather()`: turn rows and variable names into columns of data
- `spread()`: turns columns into rows and names
- `separate()`: split up one variable into multiple variables

```
require(tidyr)
```

You can find more information on this topic at
`http://garrettgman.github.io/tidying/`

# WHO TB data

The `tidyr` package includes a data frame of WHO data on new cases of TB for 212 countries over 34 years (with lots of missing data).

```r
data(who)
names(who)
```

```
 [1] "country"      "iso2"         "iso3"        
 [4] "year"         "new_sp_m014"  "new_sp_m1524"
 [7] "new_sp_m2534" "new_sp_m3544" "new_sp_m4554"
[10] "new_sp_m5564" "new_sp_m65"   "new_sp_f014" 
[13] "new_sp_f1524" "new_sp_f2534" "new_sp_f3544"
[16] "new_sp_f4554" "new_sp_f5564" "new_sp_f65"  
[19] "new_sn_m014"  "new_sn_m1524" "new_sn_m2534"
[22] "new_sn_m3544" "new_sn_m4554" "new_sn_m5564"
[25] "new_sn_m65"   "new_sn_f014"  "new_sn_f1524"
[28] "new_sn_f2534" "new_sn_f3544" "new_sn_f4554"
[31] "new_sn_f5564" "new_sn_f65"   "new_ep_m014" 
[34] "new_ep_m1524" "new_ep_m2534" "new_ep_m3544"
[37] "new_ep_m4554" "new_ep_m5564" "new_ep_m65"  
[40] "new_ep_f014"  "new_ep_f1524" "new_ep_f2534"
[43] "new_ep_f3544" "new_ep_f4554" "new_ep_f5564"
[46] "new_ep_f65"   "newrel_m014"  "newrel_m1524"
```

# The "data in names" problem

For many purposes, this is not a convenient form for the data. Here are some questions that are challenging or awkward to answer with the data as they are:

- Which countries provided data in which years (for each country and year we need to look in 56 columns to check whether any of them have data)
- How is the total number of new TB cases changing (for particular countries) over time?
- Does (reported) TB affect men and women in equal numbers?
- Which categories represent that largest fraction of new cases (in a given year and country)?

# The “data in names” problem

In particular, the variable names are storing a kind of data that we might prefer to have stored inside the data table so that we can use our data operations on it.
Each variable that begins `new` contains information on

- diagnosis method (`rel` = relapse, `sn` = negative pulmonary smear, `sp` = positive pulmonary smear, `ep` = extrapulmonary)
- sex (`f` = female, `m` = male)
- age group (`014` = 0 to 14, `1524` = 15 to 24, `2534` = 25 to 34, `3544` = 35 to 44, etc.)

# Ack: the names are not consistent

Some of the names begin `new_` and some omit that `_` and the oldest group is coded awkwardly. We could fix that here by modifying the names, but we'll do it a little later.

```r
head(names(who), 9)
```

```
[1] "country"      "iso2"         "iso3"
[4] "year"         "new_sp_m014"  "new_sp_m1524"
[7] "new_sp_m2534" "new_sp_m3544" "new_sp_m4554"
```

```r
tail(names(who), 9)
```

```
[1] "newrel_m5564" "newrel_m65"   "newrel_f014"
[4] "newrel_f1524" "newrel_f2534" "newrel_f3544"
[7] "newrel_f4554" "newrel_f5564" "newrel_f65"
```

# gather(): turn rows into columns

```r
whoLong <-
  who %>%
  gather("key", "count", 5:60)
whoLong %>% sample_n(10)
```

```
# A tibble: 10 × 6
                       country  iso2  iso3  year
                         <chr> <chr> <chr> <int>
1                        Nepal    NP   NPL  1995
2         Netherlands Antilles    AN   ANT  1984
3                    Mauritius    MU   MUS  2000
4                         Fiji    FJ   FJI  1987
5             China, Macao SAR    MO   MAC  1999
6                    Lithuania    LT   LTU  2003
7                  Saint Lucia    LC   LCA  1997
8   Iran (Islamic Republic of)    IR   IRN  1984
9         United Arab Emirates    AE   ARE  2011
10                    Malaysia    MY   MYS  1986
# ... with 2 more variables: key <chr>, count <int>
```

# separate()

```
whoLong <-
  who %>%
  gather("key", "count", 5:60) %>%
  # fix inconsistencies naming scheme
  mutate(key = sub("new[^_]", "new_", key)) %>%
  mutate(key = sub("65", "6599", key)) %>%
  separate(key, c("new", "diagnosis", "sexage"),
           sep = "_")
```

# separate(): sanity check

```
whoLong %>%
  group_by(year, diagnosis, sexage) %>%
  summarise(n_items = n()) %>%
  group_by(n_items) %>%
  summarise(n_countries = n() / (ncol(who) - 4))
```

```
# A tibble: 5 × 2
  n_items n_countries
    <int>       <dbl>
1     212          22
2     213           3
3     214           5
4     216           1
5     217           3
```

# separate(): sanity check

```
who %>% group_by(year) %>%
  summarise(n_countries = n()) %>%
  group_by(n_countries) %>%
  summarise(n_years = n())
```

```
# A tibble: 5 × 2
  n_countries n_years
        <int>   <int>
1         212      22
2         213       3
3         214       5
4         216       1
5         217       3
```

# separate(): putting it all together

```
wholong <-
  who %>%
  gather("key", "count", 5:60) %>%
  mutate(key = sub("new[^_]", "new_", key)) %>%
  mutate(key = sub("65", "6599", key)) %>%
  separate(key, c("new", "diagnosis", "sexage"),
           sep ="_") %>%
  separate(sexage, c("sex", "age"), sep = 1) %>%
  separate(age, c("age_lo", "age_hi"),
           sep = -3, remove = FALSE)
```

# separate(): putting it all together

```
whoLong %>% sample_n(5) %>% select(-country, iso2)
```

```
# A tibble: 5 × 10
   iso2  iso3  year   new diagnosis   sex   age age_lo
  <chr> <chr> <int> <chr>     <chr> <chr> <chr>  <chr>
1    HK   HKG  1984   new        ep     f  6599     65
2    AZ   AZE  1998   new        sp     m   014      0
3    TM   TKM  1988   new        sn     f   014      0
4    TN   TUN  2000   new        sn     f  4554     45
5    RU   RUS  1990   new        sn     f   014      0
# ... with 2 more variables: age_hi <chr>, count <int>
```

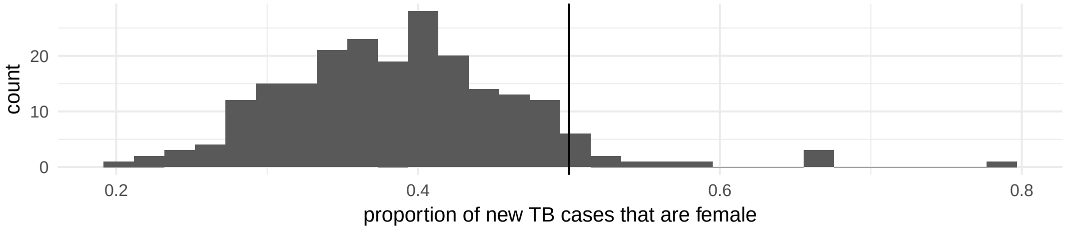
# Men vs Women

Now we can investigate one of our questions.

```
TBfemale <-
  whoLong %>%
  group_by(country, iso3, sex) %>%
  summarise(tb = sum(count, na.rm = TRUE)) %>%
  group_by(country, iso3) %>%
  mutate(total_tb = sum(tb, na.rm = TRUE)) %>%
  ungroup() %>%
  filter(sex == "f") %>%
  mutate(prop.women = tb / total_tb)
```

# Men vs Women

```
TBfemale %>%
  ggplot(aes(x = prop.women)) +
  geom_histogram() + geom_vline(xintercept = 0.5) +
  xlab("proportion of new TB cases that are female")
```



# Who are the outlier nations?

In most countries, TB is more commonly reported in men. But there are some exceptions.

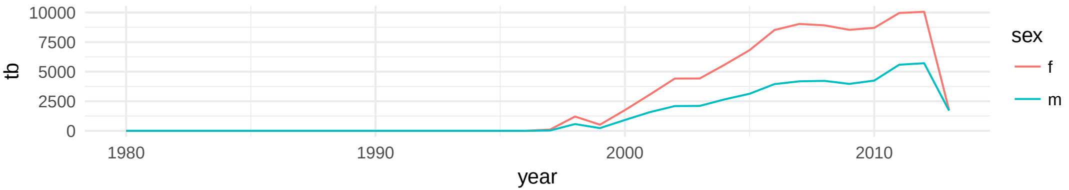
```
TBfemale %>%
  select(-country) %>%
  arrange(-prop.women) %>% head(5)
```

```
# A tibble: 5 × 5
   iso3   sex    tb total_tb prop.women
  <chr> <chr> <int>    <int>      <dbl>
1   WSM     f  1249     1607  0.7772246
2   MCO     f     2        3  0.6666667
3   SXM     f     6        9  0.6666667
4   AFG     f 93354   140225  0.6657443
5   ISL     f    75      129  0.5813953
```

# Afghanistan is interesting

The other outlier countries don't have many cases. Let's look at Afghanistan.

```
whoLong %>%
  group_by(country, iso3, year, sex) %>%
  summarise(tb = sum(count, na.rm = TRUE)) %>%
  filter(country == "Afghanistan") %>%
  ggplot() +
  geom_line(aes(x = year, y = tb, colour = sex))
```



# unite() – reverse of separate()

We can combine multiple columns into a single column with `unite()`.

```
whoLong %>% select(-new, -age) %>%
  unite(category, diagnosis, sex, age_lo, age_hi) %>%
  sample_n(4)
```

```
# A tibble: 4 × 6
      country  iso2  iso3  year   category count
        <chr> <chr> <chr> <int>      <chr> <int>
1    Maldives    MV   MDV  1999 sn_f_15_24    NA
2     Andorra    AD   AND  2002 sp_f_35_44     0
3 El Salvador    SV   SLV  1982 el_f_45_54    NA
4     Hungary    HU   HUN  2006 sn_f_45_54    76
```

# spread()

We can use spread to take columns and spread them out into rows, making our data wider. For example, we might like to have a single row that gives information about both male and female names.

```
Babynames %>%
  select(year, name, n, sex) %>%
  spread(sex, n) %>%
  sample_n(3)
```

```
# A tibble: 3 × 4
   year    name     F     M
  <dbl>   <chr> <int> <int>
1  1962 Mildred   963    NA
2  2013    Avaa     6    NA
3  1998 Sharise     9    NA
```

# Two-way names

This makes it easy to identify common "two-way names".

```
TwoWayNames <-
  Babynames %>%  select(year, name, n, sex) %>%
  spread(sex, n) %>%
  filter(F > 0, M > 0)
TwoWayNames %>%
  arrange( - pmin(F, M)) %>%
  head(4)
```

```
# A tibble: 4 × 4
   year   name      F      M
  <dbl>  <chr> <int> <int>
1  1992 Taylor 14950  8240
2  1991 Taylor 10252  7968
3  1993 Taylor 21266  7688
4  1997 Jordan  7166 14759
```

```
TwoWayNames %>%
  group_by(name) %>%
  summarise(F_total = sum(F), M_total = sum(M)) %>%
  arrange( - pmin(F_total, M_total)) %>%
  head(4)
```

# Combining data from multiple sources

```
require(nycflights13)
```

**The NYC Airlines data has (2013) data on each**

- flight [`flights`]
- airport [`airports`]
- airline [`airlines`]
- plane [`planes`]
- time/airport [`weather`]

# The NYC Airlines data

```r
names(flights)
```

```
 [1] "year"           "month"          "day"
 [4] "dep_time"       "sched_dep_time" "dep_delay"
 [7] "arr_time"       "sched_arr_time" "arr_delay"
[10] "carrier"        "flight"         "tailnum"
[13] "origin"         "dest"           "air_time"
[16] "distance"       "hour"           "minute"
[19] "time_hour"
```

```r
names(airports)
```

```
[1] "faa"  "name" "lat"  "lon"  "alt"  "tz"   "dst"
```

```r
names(airlines)
```

```
[1] "carrier" "name"
```

```r
names(weather)
```

```
 [1] "origin"     "year"       "month"
 [4] "day"        "hour"       "temp"
 [7] "dewp"       "humid"      "wind_dir"
```

# join add aircraft information to flight data

```r
flights %>% select(carrier, flight, tailnum, origin, dest) %>% head()
```

```
# A tibble: 6 × 5
  carrier flight tailnum origin  dest
    <chr>  <int>   <chr>  <chr> <chr>
1      UA   1545  N14228    EWR   IAH
2      UA   1714  N24211    LGA   IAH
3      AA   1141  N619AA    JFK   MIA
4      B6    725  N804JB    JFK   BQN
5      DL    461  N668DN    LGA   ATL
6      UA   1696  N39463    EWR   ORD
```

```r
planes %>% select(tailnum, year, model, seats) %>% head()
```

```
# A tibble: 6 × 4
  tailnum  year     model seats
    <chr> <int>     <chr> <int>
1  N10156  2004 EMB-145XR    55
2  N102UW  1998  A320-214   182
3  N103US  1999  A320-214   182
4  N104UW  1999  A320-214   182
5  N10575  2002 EMB-145LR    55
```

# x %>% join(y)

```r
# inner_join: only rows in x that have a match in y
flights %>% inner_join(planes, by="tailnum") %>% dim()
```

```
[1] 284170     27
```

```r
# left_join: all rows from x (NA's when no match)
flights %>% left_join(planes, by="tailnum") %>% dim()
```

```
[1] 336776     27
```

```r
# right_join: all rows from y (NA's when no match)
flights %>% right_join(planes, by = "tailnum") %>% dim()
```

```
[1] 284170     27
```

```r
# full_join: all rows from both x and y (NA's when no match)
flights %>% full_join(planes, by = "tailnum") %>% dim()
```

```
[1] 336776     27
```

# anti_join(): How many have no match?

```r
# anti_join: return all rows from x that have no match in y
# only columns from x are used
flights %>% anti_join(planes, by = "tailnum") %>% nrow()
```

```
[1] 52606
```

# anti_join(): What has no match?

```r
flights %>% anti_join(planes, by = "tailnum") %>%
  left_join(airlines) %>%
  group_by(carrier, name) %>%
  summarise(n = n()) %>% arrange(-n)
```

```
Joining, by = "carrier"

Source: local data frame [10 x 3]
Groups: carrier [10]

   carrier                        name     n
     <chr>                       <chr> <int>
1       MQ                   Envoy Air 25397
2       AA      American Airlines Inc. 22558
3       UA       United Air Lines Inc.  1693
4       9E           Endeavor Air Inc.  1044
5       B6             JetBlue Airways   830
6       US             US Airways Inc.   699
7       FL AirTran Airways Corporation   187
8       DL        Delta Air Lines Inc.   110
9       F9      Frontier Airlines Inc.    50
10      WN      Southwest Airlines Co.    38
```

# Exercises

1. Which is more likely to have more than a 30-minute departure delay, a smaller plane or a larger plane?
2. How many different planes flew between GRR and NYC? Which one flew most often?
3. Did any planes fly for multiple carriers during 2013?
4. Are older planes more likely to have delays?
5. How do the ages of the planes vary by carrier?
6. Are there any flights among the three NYC airports?
7. How large are the planes that make direct flights to/from GRR?
8. American Airlines had a lot of tail numbers not in `planes`. What fraction of AA flights are in this category?

# lubridate: working with dates

```r
require(lubridate)

rightnow <- now()
rightnow
```

```
[1] "2016-09-26 08:16:38 EDT"
```

```r
day(rightnow)
```

```
[1] 26
```

```r
week(rightnow)
```

```
[1] 39
```

```r
month(rightnow, label = FALSE)
```

```
[1] 9
```

```r
month(rightnow, label = TRUE)
```

```
[1] Sep
12 Levels: Jan < Feb < Mar < Apr < May < ... < Dec
```

# lubridate

Arithmetic with lubridate

```r
jan31 <- ymd("2013-01-31")
jan31 + months(0:11)
```

```
 [1] "2013-01-31" NA           "2013-03-31"
 [4] NA           "2013-05-31" NA
 [7] "2013-07-31" "2013-08-31" NA
[10] "2013-10-31" NA           "2013-12-31"
```

```r
floor_date(jan31, "month") # round down to the nearest month
```

```
[1] "2013-01-01"
```

```r
floor_date(jan31, "month") + months(0:11) + days(31)
```

```
 [1] "2013-02-01" "2013-03-04" "2013-04-01"
 [4] "2013-05-02" "2013-06-01" "2013-07-02"
 [7] "2013-08-01" "2013-09-01" "2013-10-02"
[10] "2013-11-01" "2013-12-02" "2014-01-01"
```

```r
jan31 + months(0:11) + days(31)
```

```
 [1] "2013-03-03" NA           "2013-05-01"
```

# readr::parse_number()

```r
require(readr)
parse_number("$1,200.34")
```

```
[1] 1200.34
```

```r
parse_number("-2%")
```

```
[1] -2
```

```r
# The heuristic is not perfect - it won't fail for things that
# clearly aren't numbers
parse_number("-2-2")
```

```
[1] -2
```

```r
parse_number("12abc34")
```

```
[1] 12
```

# humanparser

This also isn't perfect, but can handle a fairly wide range of names.

```r
require(humanparser)
Names <- c("Rip Van Winkle", "Fred Flintstone",
           "John Q. Public", "Oscar De La Hoya")
parse_names(Names)
```

```
  firstName    lastName          fullName middleName
1       Rip Van Winkle    Rip Van Winkle       <NA>
2      Fred Flintstone   Fred Flintstone       <NA>
3      John      Public    John Q. Public         Q.
4     Oscar De La Hoya Oscar De La Hoya       <NA>
```